

Docker 很占用空间，每当我们运行容器、拉取镜像、部署应用、构建自己的镜像时，我们的磁盘空间会被大量占用。

如果你也被这个问题所困扰，咱们就一起看一下 Docker 是如何使用磁盘空间的，以及如何回收。

docker 占用的空间可以通过下面的命令查看：

```
$ docker system df
```

```
5. luc@saturn: ~ (bash)
luc@saturn:~$ docker system df
TYPE                TOTAL          ACTIVE        SIZE          RECLAIMABLE
Images              39             4            7.459GB      6.394GB (85%)
Containers          5             1            539.9kB      3.112kB (0%)
Local Volumes       100           0            25.82GB      25.82GB (100%)
Build Cache         0             0             0B           0B
```

TYPE 列出了 docker 使用磁盘的 4 种类型：

- **Images**：所有镜像占用的空间，包括拉取下来的镜像，和本地构建的。
- **Containers**：运行的容器占用的空间，表示每个容器的读写层的空间。
- **Local Volumes**：容器挂载本地数据卷的空间。
- **Build Cache**：镜像构建过程中产生的缓存空间（只有在使用 BuildKit 时才有，Docker 18.09 以后可用）。

最后的 **RECLAIMABLE** 是可回收大小。

下面就分别了解一下这几个类型。

容器的磁盘占用

每次创建一个容器时，都会有一些文件和目录被创建，例如：

- `/var/lib/docker/containers/ID` 目录，如果容器使用了默认的日志模式，他的所有日志都会以 JSON 形式保存到此目录下。
- `/var/lib/docker/overlay2` 目录下含有容器的读写层，如果容器使用自己的文件系统保存了数据，那么就会写到此目录下。

现在我们从一个完全干净的系统开始，假设 docker 刚刚安装：

```
$ docker system df
```

TYPE	TOTAL	ACTIVE	SIZE	RECLAIMABLE
Images	0	0	0B	0B
Containers	0	0	0B	0B
Local Volumes	0	0	0B	0B
Build Cache	0	0	0B	0B

首先，我们启动一个 NGINX 容器：

```
$ docker container run --name www -d -p 8000:80 nginx:1.16
```

现在运行 `df` 命令后，就会看到：

- 一个镜像，126MB
- 一个容器

```
$ docker system df
```

TYPE	TOTAL	ACTIVE	SIZE	RECLAIMABLE
Images	1	1	126M	0B (0%)
Containers	1	1	2B	0B (0%)
Local Volumes	0	0	0B	0B
Build Cache	0	0	0B	0B

此时没有可回收空间，因为容器在运行，镜像正被使用。

现在，我们在容器内创建一个 100MB 的空文件：

```
$ docker exec -ti www \
  dd if=/dev/zero of=test.img bs=1024 count=0 seek=${1024*100}
```

```
$ docker exec -ti www \
  dd if=/dev/zero of=test.img bs=1024 count=0 seek=${1024*100}
```

再次查看空间：

```
$ docker system df
```

TYPE	TOTAL	ACTIVE	SIZE	RECLAIMABLE
Images	1	1	126M	0B (0%)
Containers	1	1	104.9MB	0B (0%)
Local Volumes	0	0	0B	0B
Build Cache	0	0	0B	0B

可以看到容器占用的空间增加了，这个文件保存在本机哪里呢？

```
$ find /var/lib/docker -type f -name test.img
/var/lib/docker/overlay2/83f177...630078/merged/test.img
/var/lib/docker/overlay2/83f177...630078/diff/test.img
```

和上面说的一样，是保存在容器的读写层。

当停止容器后，容器占用的空间就会变为可回收的：



```
$ docker stop www
# Visualizing the impact on the disk usage
$ docker system df
```

TYPE	TOTAL	ACTIVE	SIZE	RECLAIMABLE
Images	1	1	126M	0B (0%)
Containers	1	0	104.9MB	104.9MB (100%)
Local Volumes	0	0	0B	0B
Build Cache	0	0	0B	0B

如何回收呢？删除容器时会删除其关联的读写层占用的空间。

也可以一键删除所有已经停止的容器：

```
$ docker container prune
```



```
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
5e7f8e5097ace9ef5518ebf0c6fc2062ff024efb495f11ccc89df21ec9b4dcc2
Total reclaimed space: 104.9MB
```

删除容器后，镜像也可以回收了：



```
$ docker system df
```

TYPE	TOTAL	ACTIVE	SIZE	RECLAIMABLE
Images	1	0	126M	126M (100%)
Containers	0	0	0B	0B
Local Volumes	0	0	0B	0B
Build Cache	0	0	0B	0B

上面的 `docker container prune` 命令是删除停止的容器，如果想删除所有容器（包括停止的、正在运行的），可以使用下面这2个命令：

```
$ docker rm -f $(docker ps -aq)

$ docker container rm -f $(docker container ls -aq)
```

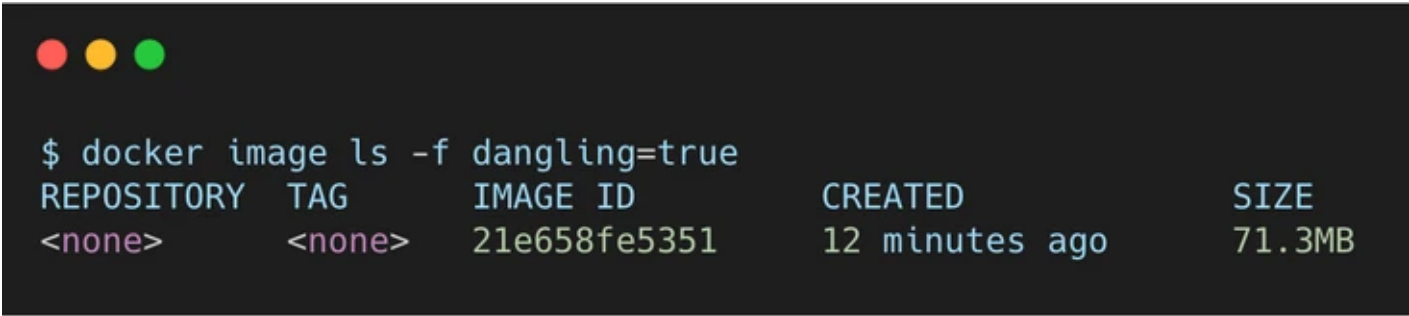
镜像的磁盘占用

有一些镜像是隐形的：

- 子镜像，就是被其他镜像引用的中间镜像，不能被删除。
- 悬挂状态的镜像，就是不会再被使用的镜像，可以被删除。

下面的命令列出所有悬挂状态的镜像：

```
$ docker image ls -f dangling=true
```



```
$ docker image ls -f dangling=true
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	21e658fe5351	12 minutes ago	71.3MB

删除这类镜像：

```
$ docker image rm $(docker image ls -f dangling=true -q)
```

或者：

```
$ docker image prune
```



```
$ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Deleted Images:
deleted: sha256:143407a3cb7efa6e95761b8cd6cea25e3f41455be6d5e7cda
deleted: sha256:738010bda9dd34896bac9bbc77b2d60add7738ad1a95e5cc
deleted: sha256:fa4f0194a1eb829523ecf3bad04b4a7bdce089c8361e2c347
deleted: sha256:c5041938bcb46f78bf2f2a7f0a0df0eea74c4555097cc9197
deleted: sha256:5945bb6e12888cf320828e0fd00728947104da82e3eb4452f
Total reclaimed space: 12.9kB
```

如果想删除所有镜像，可以使用下面的命令：

```
$ docker image rm $(docker image ls -q)
```

注意，正在被容器使用的镜像是不能被删除的。

数据卷的磁盘占用

数据卷是容器自身文件体统之外的数据存储。

例如容器中的应用有上传图片的功能，上传之后肯定不能保存在容器内部，因为容器内部的数据会随着容器的死掉而被删除，所以，这些图片要保存在容器之外，也就是数据卷。

比如我们运行了一个 MongoDB 容器做测试，导入了很多测试数据，这些数据就不是在容器内部的，是在数据卷中，因为 MongoDB 的 Dockerfile 中使用了数据卷。

测试完成后，删除了这个 MongoDB 容器，但测试数据还在，没被删除。

删除不再使用的数据卷：

```
$ docker volume rm $(docker volume ls -q)
```

或者：

```
$ docker volume prune
```




```
$ docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
d50b6402eb75d09ec17a5f57df4ed7b520c448429f70725fc5707334e5ded4d5
...

Total reclaimed space: 25.82GB
```

Build Cache 的磁盘占用

Docker 18.09 引入了 **BuildKit**，提升了构建过程的性能、安全、存储管理等能力。

删除 build cache 可以使用命令：

```
$ docker builder prune
```



```
$ docker builder prune
WARNING! This will remove all dangling build cache.
Are you sure you want to continue? [y/N] y
Deleted build cache objects:
rffq7b06h9t09xe584rn4f91e
ztexgsz949ci8mx8p5tzgdzhe
3z9jeoqbbmj3eftltawvkiayi
Total reclaimed space: 8.949kB
```

一键清理

通过上面的说明，我们知道了像容器、镜像、数据卷都提供了 **prune** 这个子命令，帮助我们回收空间。

其实，docker 系统层面也有 **prune** 这个子命令，可以一键清理没用的空间：

```
$ docker system prune
```



```
$ docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache
Are you sure you want to continue? [y/N]
```

定期执行这个命令是个好习惯。

翻译整理自：

<https://medium.com/better-pro...>